CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Metaprogramming: The C Preprocessor

Directives
Constants/Macros
Conditionals
Debugging
Preprocessor

Preprocessor stage

C Source File

C Compiler

Preprocessor

C Header Files

Compiler

Assembly Source File

Maxwell James Dunne
Preprocessor
Operation of

- Preprocessor operates on all sources files before they're pass to the compiler
- Processes special *preprocessor directives* specified in the code
- Final text of the source file after all preprocessor directives are processed is then compiler
Preprocessor Directives

Definition

Preprocessor Directives are parts of the code that give special instructions to the compiler. They always begin with a # at the beginning of the line, and are used to direct the compiler with a number of specific commands.

• Groups:
  – #defines: constants, macros
  – Conditionals

• Usage:
  – Code organization
  – Debugging
# Preprocessor Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define</td>
<td>Define a preprocessor macro.</td>
</tr>
<tr>
<td>#elif</td>
<td>Alternatively include some text based on the value of another expression, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#else</td>
<td>Alternatively include some text, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#endif</td>
<td>Terminate conditional text.</td>
</tr>
<tr>
<td>#error</td>
<td>Produce a compile-time error with a designated message.</td>
</tr>
<tr>
<td>#if</td>
<td>Conditionally include text, based on the value of an expression.</td>
</tr>
<tr>
<td>#ifdef</td>
<td>Conditionally include text, based on whether a macro name is defined.</td>
</tr>
<tr>
<td>#ifndef</td>
<td>Conditionally include text, based on if a name is not a defined macro.</td>
</tr>
<tr>
<td>#include</td>
<td>Insert text from another source file.</td>
</tr>
<tr>
<td>#line</td>
<td>Reset the line number for compiler output</td>
</tr>
<tr>
<td>#pragma</td>
<td>Allows for extending preprocessor directives beyond what's in the standard</td>
</tr>
<tr>
<td>#</td>
<td>Null directive</td>
</tr>
<tr>
<td>#warning</td>
<td>Emits a warning described by the rest of the line</td>
</tr>
</tbody>
</table>
Preprocessor Directives
Text substitution using `#define`

- Defines a text substitution label

**Syntax**

```
#define label text
```

- Each instance of `label` will be replaced with `text` by the preprocessor unless `label` is inside a string
- `text` is optional
- Uses no memory

**Example**

```
#define PI 3.14159
#define MOL 6.02E23
#define MCU "PIC32MX320F128H"
#define PI_2 2 * PI
#define __STDIO_H__
```
Preprocessor Directives

Text substitution using `#define`

- Labels must be valid identifiers

Example

```c
#define 0 1
#define __WRONG
#define __WRONG
#define RIGHT
```
Preprocessor Directives

Text substitution using `#define`

- Text goes until the end of the line
  - Unless newline is escaped with a `\`

Example

```c
#define true false
#define true \false
```

- Constants can be nested

Example

```c
#define OLED_NUM_LINES (OLED_DRIVER_PIXEL_ROWS \/ ASCII_FONT_HEIGHT)
```
## Preprocessor Directives

### Predefined constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FILE</strong></td>
<td>Full path of current file</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>The current line in the file</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>The current date as a string, like &quot;Jan 27 2014&quot;</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>The current time as a string, like &quot;17:20:50&quot;</td>
</tr>
<tr>
<td><strong>func</strong></td>
<td>The current function as a string, like &quot;main&quot;</td>
</tr>
<tr>
<td><strong>DEBUG</strong></td>
<td>When debugging is specified in MPLAB X, <strong>not part of the standard</strong>!</td>
</tr>
</tbody>
</table>
Preprocessor Directives

#undef

Syntax

```
#define M_PI 3.14
#undef M_PI
#define M_PI 3.141592653589793238462643383279502884197
```

- Deletes a macro definition
- Allows you to change a macro
  - Error when macros are redefined otherwise
Preprocessor Directives

Argument Macros

- Create a function-like macro

**Syntax**

```c
#define LABEL(arg1, ..., argn) code
```

- The `code` must fit on a single line or use `\` to split lines
- Text substitution used to insert arguments into `code`
- Each instance of `LABEL()` will be expanded into `code`
- This is not the same as a C function! No stack allocation.

**Example**

```c
#define MIN(x, y) ((x) < (y) ? (x) : (y))
#define SQUARE(x) ((x) * (x))
#define SWAP(x, y) { (x) ^= (y); (y) ^= (x); (x) ^= (y); }
```
Preprocessor Directives

Argument Macros – Side Effects

Example

```c
#define SQUARE(x) x * x

Extreme care must be exercised when using macros. Consider the following use of the above macro:

```c
i = 5;
a = SQUARE(i + 3);
```
Preprocessor Directives

Argument Macros – Side Effects

Example

#define SQUARE(x) ((x)*(x))

Extreme care must be exercised when using macros. Consider the following use of the above macro:

i = 5;
a = SQUARE(i++);

(32)(32) * (32)
Macros with `#define`

Argument Macros – Side Effects

Example

```c
#define ABS(x) (((x) > 0) ? (x) : (-x))
#define NORM1(x, y) (ABS((x)) + ABS((y)))

int x = NORM1(5, 6.6);
```

```c
int x = (((5) > 0)?(5):(-5)) + (((6.6) > 0)?(6.6):(-6.6));
```
Macros with \texttt{define}

Emulating functions

- Functions provide useful features:
  - Encapsulation
  - Evaluate as an expression
  - Return values
Preprocessor Directives
Emulating functions

• For encapsulation

Example

```c
#define LABEL(arg1, ..., argn) {
    ...
}
```

- Code blocks forces all code in the macro to execute in the same context
- Also allows for temporary variables within the macros
Preprocessor Directives

Emulating functions

Example

```
#define INIT() TRISA = 5; LATA = 5;

if (beginStartup)
    INIT();

    LATA = 5;
```
Preprocessor Directives

Emulating functions

Example

```c
#define INIT() {TRISA = 5; LATA = 5;};
```

```c
if (beginStartup)
   INIT();
else
   ...
```
Preprocessor Directives
Emulating functions

• For encapsulation with expression-ness

Example

```c
#define LABEL(arg1, ..., argn) do {
    ...
    } while (0)
```

- Code blocks forces all code in the macro to execute in the same context
  - Also allows for temporary variables within the macros
- `while`-statement allows for semi-colon termination
  - Generates a single statement
Preprocessor Directives

Emulating functions

• To "return" values, just have the statement evaluate to a value

Example

```c
#define LABEL(arg1, ..., argn) VALUE
```
Preprocessor Directives

Stringification of macro values

Example

```
#define VERSION 6.3
#define TEXTIFY(x) #x

printf("\%s", TEXTIFY(VERSION));
```

6.3
Preprocessor Directives
Stringification of macro values

- You need another layer of indirection

Example

```c
#define TEXTIFY(x) TEXTIFY_HELPER(x)
#define TEXTIFY_HELPER(x) #x
#define MAJOR_VER 1
#define MINOR_VER 3
#define VERSION_STRING TEXTIFY(MAJOR_VER) \ 
    "\" \ 
    TEXTIFY(MINOR_VER)

printf("%s", TEXTIFY(VERSION));
```

1.3
Preprocessor Directives

Token concatenation

- To combine argument with existing token to generate identifiers

Example

```c
#define DEBUGIFY(x) x ## _DEBUG

printf("%s", DEBUGIFY(asdf));
```
Preprocessor Directives

Conditional compilation

• Control what code actually gets compiled
  – Already seen this with header guards

Example

```c
#ifndef BUTTONS_H
#define BUTTONS_H
...
#endif
```
Preprocessor Directives
Conditional compilation

- Family of if-statements
  - `#if`
  - `#ifdef`
  - `#ifndef`
- Ended with `#endif`
- `#if` is the general case
  - `#ifdef/#ifndef` only check if a macro has been defined
Preprocessor Directives

Emulating functions

```
#if INIT
#endif
#if 0
#endif
#if defined(_WIN32)
#endif
#if defined(__unix__) && !defined(__APPLE__)
#endif
#if __STDC_VERSION__ > 199409L
#endif
```
```
d < p

# if defined (d < p)
    -> functions

# else
```
Preprocessor Directives

Conditional compilation

• `#ifdef text`
  - Same as `#if defined(...)`

• `#ifndef text`
  - Same as `#if !defined(...)`

• `#elif text`
  - Else-if, follows same rules as `#if`

• `#else`

• `#endif`
Preprocessor Directives

Unit testing

• Conditionally compile in test code

Example

```c
int main(void)
{
    // Initialization code
    #if 0
    #elif 1
    // Test code
    #endif
    // Main program
}
```
Preprocessor Directives

Fatal errors

• Output location of failure and stop running

```c
#define FATAL_ERROR()  
    do {  
        printf("FATAL ERROR at \%s:\%s():\%ld\n",  
            __FILE__, __func__, __LINE__);  
        TRISE = 0;  
        LATE = 0xFF;  
    } while (1)
```
Preprocessor Directives
Forcing compilation errors/warnings

• `#warning text`
  – Outputs compilation warning

• `#error text`
  – Outputs compilation error

Example

```c
#if __STDC_VERSION__ < 199901
#error "Must be compiled with C99 or greater"
#endif
```
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Switch statements
**switch Statement**

### Syntax

```c
switch (expression)
{
    case const-expr\_1: statements\_1
    :
    case const-expr\_n: statements\_n
    default: statements\_n+1
}
```

- **expression** is evaluated and tested for a match with the **const-expr** in each **case** clause
- The **statements** in the matching **case** clause is executed
**switch Statement**

Flow Diagram (default)

```
START

Const-exp1 = expression?
  YES  \rightarrow  statement1
  NO

Const-exp2 = expression?
  YES  \rightarrow  statement2
  NO

\vdots

Const-expn = expression?
  YES  \rightarrow  statementn
  NO

statement_{n+1}

END
```

Notice that each statement falls through to the next.

This is the default behavior of the `switch` statement.
switch Statement

Flow Diagram (modified)

START

Const-expr\textsubscript{1} = expression? YES → statement\textsubscript{1} break; NO

Const-expr\textsubscript{2} = expression? YES → statement\textsubscript{2} break; NO

\ldots

Const-expr\textsubscript{n} = expression? YES → statement\textsubscript{n} break; NO

statement\textsubscript{n+1}

END

Adding a break statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed.
switch Statement
Simple example

switch Example 1

switch (channel) {
    case 2:  puts("WBBM Chicago");  break;
    case 3:  puts("DVD Player");    break;
    case 4:  puts("WTMJ Milwaukee"); break;
    case 5:  puts("WMAQ Chicago");  break;
    case 6:  puts("WITI Milwaukee"); break;
    case 7:  puts("WLS Chicago");   break;
    case 9:  puts("WGN Chicago");   break;
    case 10: puts("WMVS Milwaukee"); break;
    case 11: puts("WTTW Chicago");  break;
    case 12: puts("WISN Milwaukee"); break;
    default: puts("No Signal Available");
}

Maxwell James Dunne
switch Statement

Styling

switch Example 1

```c
switch (channel) {
    case 2:
        puts("WBBM Chicago");
        break;
    case 3:
        puts("DVD Player");
        break;
    case 4:
        puts("WTMJ Milwaukee");
        break;
    ...
}
```
be a ... 
be a
or a
switch Statement

With ASCII

switch Example 2

```c
switch (letter) {
    case 'a':
        puts("Letter 'a' found.");
        break;
    case 'b':
        puts("Letter 'b' found.");
        break;
    case 'c':
        puts("Letter 'c' found.");
        break;
    default:
        puts("Letter not in list.");
}
```
switch Statement
Fall-through

switch Example 3

```c
switch (channel) {
    case 4:
    case 5:
    case 6:
    case 7:
        puts("VHF Station");
        break;
    case 9:
    case 10:
    case 11:
    case 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```

Maxwell James Dunne
switch Statement

Range syntax

switch Example 3

```c
switch(channel) {
    case 4 ... 7:
        puts("VHF Station");
        break;
    case 9 ... 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```
switch Statement

Real-world example

switch Example 2

```c
bool IsHex(char character)
{
    switch (character) {
    case 'a' ... 'f':
    case 'A' ... 'F':
    case '0' ... '9':
        return true;
    default:
        return false;
    }
}
```
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
State machines
State machines

- Known as Finite State Machines (FSM)
- Mathematical model of computation where system has a single state
- Triggering conditions can change that state
- FSMs are defined completely by both their states and the transitions between them
State machines

State

- The system only exists in one state at a time
- State persists through time
- Certain conditions can change the state to another state
  - These are specific to the current state
State machines

Transitions

• Events trigger transitions between states
• A combination of events can be used
• Transitions are all mutually exclusive
• At any given time there must be a valid transition for a state
  – If no transition is explicitly stated, an implied loopback transition exists
State machines

Benefits

- Provides a formal way to reason about a system
  - Allows for testing before writing any code
- Can be easily visualized
- Are language independent
- States are only dependent on current state and current inputs
State machines

When to use

- Can be used whenever there are a finite set of states for the system
  - Car transmission
  - Stoplight
  - Vending machine
  - Toaster oven
  - Video games
State machines

Use in the SeaSlug

• Transmission protocol
  – Mission management
  – Parameter management

• Operating state
  – Handling errors/system faults

• Calibration
  – Rudder
  – Radio controller
State machines

Diagrams

STATE_1

condition1
action1

STATE_2

condition2
action2

action 0

C³ₐ₃
typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
{
    switch (state) {
    case STATE_1:
    default:
        if (condition1) {
            Action1();
            state = STATE_2;
        }
    break;
    case STATE_2:
        if (condition2) {
            Action2();
            state = STATE_1;
        }
    }
typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
int main (void) {
    // Initialize system

    // Event loop
    while (1) {
        // State machine
        switch (state) {
            ...   
        }
    }
}